Amendments to the Specification:

A substitute specification is provided which replaces all instances of the word "Java" with "Java™".

Please amend the abstract of the specification as follows:

Improved techniques for representation of objects in a Java™ programming environment are disclosed. The techniques are highly suitable for representation of Java™ objects inside virtual machines, especially those that operate with limited resources (e.g., embedded systems). In accordance with one embodiment, a Java™ object representation is disclosed. As will be appreciated, the Java™ object representation provides a reference that can be used to directly access the internal class representation associated with the object. The internal class representation provides information regarding the Java™ object (e.g., object size, object type, static fields, etc.) As a result, information regarding Java™ objects can quickly be accessed. This means that the processing time conventionally needed to access information regarding Java™ objects is reduced. Thus, performance of virtual machines, especially in systems with limited computing power and/or memory, can be enhanced.

TWO TIER CLUSTERS FOR REPRESENTATION OF OBJECTS IN JAVA™ PROGRAMMING ENVIRONMENTS

BACKGROUND OF THE INVENTION

The present invention relates generally to Java™ programming environments, and more particularly, to techniques suitable for representation of objects in a Java™ programming environment.

One of the goals of high level languages is to provide a portable programming environment such that the computer programs may easily be ported to another computer platform. High level languages such as "C" provide a level of abstraction from the underlying computer architecture and their success is well evidenced from the fact that most computer applications are now written in a high level language.

Portability has been taken to new heights with the advent of the World Wide Web ("the Web") which is an interface protocol for the Internet which allows communication between diverse computer platforms through a graphical interface. Computers communicating over the Web are able to download and execute small applications called applets. Given that applets may be executed on a diverse assortment of computer platforms, the applets are typically executed by a Java™ virtual machine.

Recently, the Java™ programming environment has become quite popular. The Java™ programming language is a language that is designed to be portable enough to be executed on a wide range of computers ranging from small devices (e.g., pagers, cell phones and smart cards) up to supercomputers. Computer programs written in the Java™ programming language (and other languages) may be compiled into Java™ Bytecode instructions that are suitable for execution by a Java™ virtual machine implementation. The Java™ virtual machine is commonly implemented in software by means of an interpreter for the Java™ virtual machine instruction set but, in general, may be software, hardware, or both. A particular Java™ virtual machine implementation and corresponding support libraries together constitute a Java™ runtime environment.

Computer programs in the Java™ programming language are arranged in one or more classes or interfaces (referred to herein jointly as classes or class files). Such programs are generally platform, i.e., hardware and operating system, independent. As such, these computer programs may be executed without modification on any computer that is able to run an implementation of the Java™ runtime environment.

Object-oriented classes written in the Java™ programming language are compiled to a particular binary format called the "class file format." The class file includes various components associated with a single class. These components can be, for example, methods and/or interfaces associated with the class. In addition, the class file format can include a significant amount of ancillary information that is associated with the class. The class file format (as well as the general operation of the Java™ virtual machine) is described in some detail in The Java™ Virtual Machine Specification, Second Edition, by Tim Lindholm and Frank Yellin, which is hereby incorporated herein by reference.

Fig. 1A shows a progression of a simple piece of a Java™ source code 101 through execution by an interpreter, the Java™ virtual machine. The Java™ source code 101 includes the classic Hello World program written in Java™. The source code is then input into a Bytecode compiler 103 that compiles the source code into Bytecodes. The Bytecodes are virtual machine instructions as they will be executed by a software emulated computer. Typically, virtual machine instructions are generic (*i.e.*, not designed for any specific microprocessor or computer architecture) but this is not required. The Bytecode compiler outputs a Java™ class file 105 that includes the Bytecodes for the Java™ program. The Java™ class file is input into a Java™ virtual machine 107. The Java™ virtual machine is an interpreter that decodes and executes the Bytecodes in the Java™ class file. The Java™ virtual machine is an interpreter, but is commonly referred to as a virtual machine as it emulates a microprocessor or computer architecture in software (e.g., the microprocessor or computer architecture may not exist in hardware).

Fig. 1B illustrates a simplified class file 100. As shown in Fig. 1B, the class file 100 includes a constant pool 102 portion, interfaces portion 104,

fields portion 106, methods portion 108, and attributes portion 110. The attributes (or attributes table) 110 portion represents the attributes associated with the class file 100. This allows for one or more attributes to be defined, each of which can be associated with one or more components of the class file. As is known to those skilled in the art, the Java™ virtual machine implementations are allowed to define and use various attributes. In addition, the virtual machine's implementations ignore attributes that they do not recognize. Thus, a class file may contain one or more attributes, all or none of which may be recognized by a particular virtual machine implementation.

Conventionally, Java™ objects are represented in memory so that the methods associated with the objects can be referenced from the object representation. Typically, there is a reference from the Java™ object representation directly to a method table that includes the methods associated with the object. Although the direct reference to the method table allows method invocations to be performed, the conventional object representation in Java™ requires some processing to find information about the object (e.g., object type, object size, static fields, etc.) Such information about the Java™ object can be stored in the internal class representation of the object. In other words, the virtual machine typically internally represents and stores the information associated with the Java™ object's class. However, accessing this information takes up valuable processing time. This can seriously hinder performance of virtual machines, especially in systems with limited computing power and/or memory.

Furthermore, using conventional Java™ object representations, it is difficult to implement a single "garbage collection" scheme that would allow removal of Java™ objects, as well as Java™ classes. In other words, conventionally, one garbage collection method is used to remove Java™ objects when they are no longer needed, and another garbage collection method is used to remove classes from memory when there are no longer needed. Thus, conventionally, garbage collection can use a significant amount of memory and computing time of conventional virtual machines. As a result, the performance of virtual machines can be adversely affected.

especially those operating with relatively smaller resources (e.g., embedded systems)

In view of the foregoing, improved techniques for representation of objects in Java™ programming environments are needed.

5

Fig. 4 illustrates a method for accessing information regarding a Java™ object using an object representation, in accordance with one embodiment of the invention.

DETAILED DESCRIPTION OF THE INVENTION

As noted in the background, typically, the virtual machines internally represent and store the information associated with the Java™ object's class. However, accessing this information using conventional techniques takes up valuable processing time. This can seriously hinder performance of virtual machines, especially in systems with limited computing power and/or memory.

The present invention pertains to techniques for representation of objects in a Java™ programming environment. The techniques are highly suitable for representation of Java™ objects inside virtual machines, especially those that operate with limited resources (e.g., embedded systems). In accordance with one aspect of the invention,

a Java™ object representation is disclosed. As will be appreciated, the Java™ object representation provides a reference that can be used to directly access the internal class representation associated with the object. The internal class representation provides information regarding the Java™ object (e.g., object size, object type, static fields, etc.) As a result, the invention allows quick access to information regarding Java™ objects. This means that the processing time conventionally needed to access information regarding Java™ objects is reduced. Thus, the invention can enhance performance of virtual machines, especially in systems with limited computing power and/or memory.

Embodiments of the invention are discussed below with reference to Figs. 2-4. However, those skilled in the art will readily appreciate that the detailed description given herein with respect to these figures is for explanatory purposes only as the invention extends beyond these limited embodiments.

Fig. 2 represents a Java™ computing environment including a cluster 200 in accordance with one embodiment of the invention. The cluster 200 is suitable for implementation in a memory portion of a Java™ virtual machine. As shown in Fig. 2, the cluster 200 includes a plurality of two-tier object representations 202, 204, 206 and 208, wherein each of the object representations consist of a first portion C and second portion O. The first portions C_1, C_2, C_i, and C_n represent references to Java™ class. The second portions O_1, O_2, O_i, and O_n represent references to Java™ object. Accordingly, each of the two-tier object representations 202, 204, 206 and 208 provide references to a Java™ class and an object associated with that class. For example, the Two-tier object representation 206 provides the reference C_i to class I and reference O_i to the object I of class I. As will be appreciated by those skilled in the art, the arrangement of the cluster 200 allows for efficient access partly because the first and second portions can be of the same size (e.g., 4 bytes).

The reference to classes can be a reference to an internal class representation of the class. Fig. 3 illustrates a Java™ object representation 300 in accordance with one embodiment of the invention. The Java™ object representation 300 illustrates in greater detail the two-tier object representations 202, 204, 206 and 208 of Fig. 2. As shown in Fig. 3, the object representation 300 includes a first reference 302 to an internal class representation 304. The internal class representation 304 provides information regarding the Java™ object. This information can include, for example, a method table 306 and a field descriptor table 308, as well as other information relating to the Java™ object. In the described embodiment, the method table 310 immediately follows a header 309 which is of a predetermined size.

As will be appreciated, the first reference 302 can be used to directly access the internal class representation 304 so that information regarding the Java™ object can be accessed quickly. As a result, information regarding objects can be accessed more quickly than conventional techniques which require more processing to find this information.

In addition, the object representation 300 includes a second reference 310 to instance fields associated with the Java™ object. These instance fields can be unique for each object and can, for example, include instance variables I_1- I_n associated with the Java™ object. Instance fields in the context of the Java™ programming language are well known to those skilled in the art.

It should be noted that the internal object representation 300 may include an identifier that uniquely identifies the Java™ object. As will be appreciated by those skilled in the art, the identifier can be a hash key. In one embodiment, the address of the first reference 302 is used as the hash key. It should also be noted that the first and second references 232 and 310 represent two consecutive memory addresses. As such, each of the first and second references 302 and 310 can be four consecutive bytes (one word) in a memory portion of the virtual machine.

Fig. 4 illustrates a method 400 for identifying active Java™ objects and classes in accordance with one embodiment of the invention. As such, the method 400 can be used in a virtual machine to perform garbage collection. Moreover, the method 400 can be used to perform garbage collection for both Java™ classes and objects. Accordingly, the method 400 allows implementation of efficient garbage collection applications.

Initially, at operation 402, a sequential read of a cluster of two-tier Java™ object representations is initiated. Next, at operation 404, a determination is made as to whether Java™ objects or Java™ classes are to be identified. If it is determined at operation 404 that Java™ objects are to be identified, the method 400 proceeds to operation 406 where references to Java™ objects are sequentially read from the cluster of two-tier Java™ object representations. Thereafter, at operation 418, the memory addresses that have been read are marked. The method 400 ends following operation 408. However, if it is determined at operation 404 that Java™ objects are to be identified, the method 400 proceeds to operation 410 where references to Java™ classes are sequentially read from the cluster of two-tier Java™ object representations. Thereafter, at operation 408, the memory addresses that have been read are marked. The method 400 ends following operation 408.

The many features and advantages of the present invention are apparent from the written description, and thus, it is intended by the appended claims to cover all such features and advantages of the invention. Further, since numerous modifications and changes will readily occur to those skilled in the art, it is not desired to limit the invention to the exact construction and operation as illustrated and described. Hence, all suitable modifications and equivalents may be resorted to as falling within the scope of the invention.

SUMMARY OF THE INVENTION

Broadly speaking, the present invention relates to techniques for representation of objects in a Java™ programming environment. The techniques are highly suitable for representation of Java™ objects inside virtual machines, especially those that operate with limited resources (e.g., embedded systems). In accordance with one aspect of the invention, a Java™ object representation is disclosed. As will be appreciated, the Java™ object representation provides a reference that can be used to directly access the internal class representation associated with the object. The internal class representation provides information regarding the Java™ object (e.g., object size, object type, static fields, etc.) As a result, the invention allows quick access to information regarding Java™ objects. This means that the processing time conventionally needed to access information regarding Java™ objects is reduced. Thus, the invention can enhance performance of virtual machines, especially in systems with limited computing power and/or memory.

The invention can be implemented in numerous ways, including as a method, an apparatus, a computer readable medium, and a database system. Several embodiments of the invention are discussed below.

As a Java™ object representation suitable for use by a Java™ virtual machine, one embodiment of the invention includes a first reference to an internal class representation of the Java™ object, a second reference to instance fields associated with the Java™ object. The first reference is a direct reference to the internal class representation of the Java™ object.

As a method for representing a Java™ object in a virtual machine, one embodiment of the invention includes the acts of: allocating a first reference in a memory portion of the virtual machine, wherein the first reference is a reference to an internal class representation of the Java™ object; and allocating a second reference in a memory portion of the virtual machine, wherein the second reference is a reference to instance fields associated with

the Java™ object; and wherein the first reference is a direct reference to the internal class representation of the Java™ object.

As a method of accessing information regarding a Java™ object, one embodiment of the invention includes the acts of identifying an object representation associated with the Java™ object; using a first reference in the object representation to locate an appropriate internal class representation associated with the Java™ object; accessing information regarding the Java™ object from the internal class representation; and wherein the object is represented in a Java™ virtual machine.

As a computer readable media including computer program code for a Java™ object representation suitable for use by a Java™ virtual machine, one embodiment of the invention includes computer program code for a first reference to an internal class representation of the Java™ object; computer program code for a second reference to instance fields associated with the Java™ object. The first reference is a direct reference to the internal class representation of the Java™ object.

These and other aspects and advantages of the present invention will become more apparent when the detailed description below is read in conjunction with the accompanying drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

The present invention will be readily understood by the following detailed description in conjunction with the accompanying drawings, wherein like reference numerals designate like structural elements, and in which:

Fig. 1A shows a progression of a simple piece of a Java™ source code through execution by an interpreter, the Java™ virtual machine.

Fig. 1B illustrates a simplified class file.

Fig. 2 represents a Java™ computing environment including a Java™ object representation in accordance with one embodiment of the invention.

Fig 3 illustrates a method for representing Java™ objects in a Java™ computing environment.

Fig. 4 illustrates a method for accessing information regarding a Java™ object using an object representation, in accordance with one embodiment of the invention.

DETAILED DESCRIPTION OF THE INVENTION

As noted in the background, typically, the virtual machines internally represent and store the information associated with the Java™ object's class. However, accessing this information using conventional techniques takes up valuable processing time. This can seriously hinder performance of virtual machines, especially in systems with limited computing power and/or memory.

The present invention pertains to techniques for representation of objects in a Java™ programming environment. The techniques are highly suitable for representation of Java™ objects inside virtual machines, especially those that operate with limited resources (e.g., embedded systems). In accordance with one aspect of the invention, a Java™ object representation is disclosed. As will be appreciated, the Java™ object representation provides a reference that can be used to directly access the internal class representation associated with the object. The internal class representation provides information regarding the Java™ object (e.g., object size, object type, static fields, etc.) As a result, the invention allows quick access to information regarding Java™ objects. This means that the processing time conventionally needed to access information regarding Java™ objects is reduced. Thus, the invention can enhance performance of virtual machines, especially in systems with limited computing power and/or memory.

Embodiments of the invention are discussed below with reference to Figs. 2-4. However, those skilled in the art will readily appreciate that the detailed description given herein with respect to these figures is for explanatory purposes only as the invention extends beyond these limited embodiments.

Fig. 2 represents a Java™ computing environment including a cluster 200 in accordance with one embodiment of the invention. The cluster 200 is suitable for implementation in a memory portion of a Java™ virtual machine. As shown in Fig. 2, the cluster 200 includes a plurality of two-tier object representations 202, 204, 206 and 208, wherein each of the object representations consist of a first portion C and second portion O. The first portions C_1, C_2, C_i, and C_n represent references to Java™ class. The second portions O_1, O_2, O_i, and O_n represent references to Java™ object. Accordingly, each of the two-tier object representations 202, 204, 206 and 208 provide references to a Java™ class and an object associated with that class. For example, the Two-tier object representation 206 provides the reference C_i to class I and reference O_i to the object I of class I. As will be appreciated by those skilled in the art, the arrangement of the cluster 200 allows for efficient access partly because the first and second portions can be of the same size (e.g., 4 bytes).

The reference to classes can be a reference to an internal class representation of the class. Fig. 3 illustrates a Java™ object representation 300 in accordance with one embodiment of the invention. The Java™ object representation 300 illustrates in greater detail the two-tier object representations 202, 204, 206 and 208 of Fig. 2. As shown in Fig. 3, the object representation 300 includes a first reference 302 to an internal class representation 304. The internal class representation 304 provides information regarding the Java™ object. This information can include, for example, a method table 306 and a field descriptor table 308, as well as other information relating to the Java™ object. In the described embodiment, the method table 310 immediately follows a header 309 which is of a predetermined size.

As will be appreciated, the first reference 302 can be used to directly access the internal class representation 304 so that information regarding the Java™ object can be accessed quickly. As a result, information regarding objects can be accessed more quickly than conventional techniques which require more processing to find this information.

In addition, the object representation 300 includes a second reference 310 to instance fields associated with the Java™ object. These instance fields can be unique for each object and can, for example, include instance variables I_1- I_n associated with the Java™ object. Instance fields in the context of the Java™ programming language are well known to those skilled in the art.

It should be noted that the internal object representation 300 may include an identifier that uniquely identifies the Java™ object. As will be appreciated by those skilled in the art, the identifier can be a hash key. In one embodiment, the address of the first reference 302 is used as the hash key. It should also be noted that the first and second references 232 and 310 represent two consecutive memory addresses. As such, each of the first and second references 302 and 310 can be four consecutive bytes (one word) in a memory portion of the virtual machine.

Fig. 4 illustrates a method 400 for identifying active Java™ objects and classes in accordance with one embodiment of the invention. As such, the method 400 can be used in a virtual machine to perform garbage collection. Moreover, the method 400 can be used to perform garbage collection for both Java™ classes and objects. Accordingly, the method 400 allows implementation of efficient garbage collection applications.

Initially, at operation 402, a sequential read of a cluster of two-tier Java™ object representations is initiated. Next, at operation 404, a determination is made as to whether Java™ objects or Java™ classes are to be identified. If it is determined at operation 404 that Java™ objects are to be identified, the method 400 proceeds to operation 406 where references to Java™ objects are sequentially read from the cluster of two-tier Java™ object representations. Thereafter, at operation 418, the memory addresses that have been read are marked. The method 400 ends following operation 408. However, if it is determined at operation 404 that Java™ objects are to be identified, the method 400 proceeds to operation 410 where references to Java™ classes are sequentially read from the cluster of two-tier Java™ object representations. Thereafter, at operation 408, the memory addresses that have been read are marked. The method 400 ends following operation 408.

The many features and advantages of the present invention are apparent from the written description, and thus, it is intended by the appended claims to cover all such features and advantages of the invention. Further, since numerous modifications and changes will readily occur to those skilled in the art, it is not desired to limit the invention to the exact construction and operation as illustrated and described. Hence, all suitable modifications and equivalents may be resorted to as falling within the scope of the invention.

TWO TIER CLUSTERS FOR REPRESENTATION OF OBJECTS IN JAVA™ PROGRAMMING ENVIRONMENTS

BACKGROUND OF THE INVENTION

The present invention relates generally to Java™ programming environments, and more particularly, to techniques suitable for representation of objects in a Java™ programming environment.

One of the goals of high level languages is to provide a portable programming environment such that the computer programs may easily be ported to another computer platform. High level languages such as "C" provide a level of abstraction from the underlying computer architecture and their success is well evidenced from the fact that most computer applications are now written in a high level language.

Portability has been taken to new heights with the advent of the World Wide Web ("the Web") which is an interface protocol for the Internet which allows communication between diverse computer platforms through a graphical interface. Computers communicating over the Web are able to download and execute small applications called applets. Given that applets may be executed on a diverse assortment of computer platforms, the applets are typically executed by a Java™ virtual machine.

Recently, the Java™ programming environment has become quite popular. The Java™ programming language is a language that is designed to be portable enough to be executed on a wide range of computers ranging from small devices (e.g., pagers, cell phones and smart cards) up to supercomputers. Computer programs written in the Java™ programming language (and other languages) may be compiled into Java™ Bytecode instructions that are suitable for execution by a Java™ virtual machine implementation. The Java™ virtual machine is commonly implemented in software by means of an interpreter for the Java™ virtual machine instruction set but, in general, may be software, hardware, or both. A particular Java™ virtual machine implementation and corresponding support libraries together constitute a Java™ runtime environment.

Computer programs in the Java™ programming language are arranged in one or more classes or interfaces (referred to herein jointly as classes or class files). Such programs are generally platform, i.e., hardware and operating system, independent. As such, these computer programs may be executed without modification on any computer that is able to run an implementation of the Java™ runtime environment.

Object-oriented classes written in the Java™ programming language are compiled to a particular binary format called the "class file format." The class file includes various components associated with a single class. These components can be, for example, methods and/or interfaces associated with the class. In addition, the class file format can include a significant amount of ancillary information that is associated with the class. The class file format (as well as the general operation of the Java™ virtual machine) is described in some detail in The Java™ Virtual Machine Specification, Second Edition, by Tim Lindholm and Frank Yellin, which is hereby incorporated herein by reference.

Fig. 1A shows a progression of a simple piece of a Java™ source code 101 through execution by an interpreter, the Java™ virtual machine. The Java™ source code 101 includes the classic Hello World program written in Java™. The source code is then input into a Bytecode compiler 103 that compiles the source code into Bytecodes. The Bytecodes are virtual machine instructions as they will be executed by a software emulated computer. Typically, virtual machine instructions are generic (*i.e.*, not designed for any specific microprocessor or computer architecture) but this is not required. The Bytecode compiler outputs a Java™ class file 105 that includes the Bytecodes for the Java™ program. The Java™ class file is input into a Java™ virtual machine 107. The Java™ virtual machine is an interpreter that decodes and executes the Bytecodes in the Java™ class file. The Java™ virtual machine is an interpreter, but is commonly referred to as a virtual machine as it emulates a microprocessor or computer architecture in software (e.g., the microprocessor or computer architecture may not exist in hardware).

Fig. 1B illustrates a simplified class file 100. As shown in Fig. 1B, the class file 100 includes a constant pool 102 portion, interfaces portion 104,

fields portion 106, methods portion 108, and attributes portion 110. The attributes (or attributes table) 110 portion represents the attributes associated with the class file 100. This allows for one or more attributes to be defined, each of which can be associated with one or more components of the class file. As is known to those skilled in the art, the Java™ virtual machine implementations are allowed to define and use various attributes. In addition, the virtual machine's implementations ignore attributes that they do not recognize. Thus, a class file may contain one or more attributes, all or none of which may be recognized by a particular virtual machine implementation.

Conventionally, Java™ objects are represented in memory so that the methods associated with the objects can be referenced from the object representation. Typically, there is a reference from the Java™ object representation directly to a method table that includes the methods associated with the object. Although the direct reference to the method table allows method invocations to be performed, the conventional object representation in Java™ requires some processing to find information about the object (e.g., object type, object size, static fields, etc.) Such information about the Java™ object can be stored in the internal class representation of the object. In other words, the virtual machine typically internally represents and stores the information associated with the Java™ object's class. However, accessing this information takes up valuable processing time. This can seriously hinder performance of virtual machines, especially in systems with limited computing power and/or memory.

Furthermore, using conventional Java™ object representations, it is difficult to implement a single "garbage collection" scheme that would allow removal of Java™ objects, as well as Java™ classes. In other words, conventionally, one garbage collection method is used to remove Java™ objects when they are no longer needed, and another garbage collection method is used to remove classes from memory when there are no longer needed. Thus, conventionally, garbage collection can use a significant amount of memory and computing time of conventional virtual machines. As a result, the performance of virtual machines can be adversely affected,

especially those operating with relatively smaller resources (e.g., embedded systems)

In view of the foregoing, improved techniques for representation of objects in Java™ programming environments are needed.

5

SUMMARY OF THE INVENTION

Broadly speaking, the present invention relates to techniques for representation of objects in a Java™ programming environment. The techniques are highly suitable for representation of Java™ objects inside virtual machines, especially those that operate with limited resources (e.g., embedded systems). In accordance with one aspect of the invention, a Java™ object representation is disclosed. As will be appreciated, the Java™ object representation provides a reference that can be used to directly access the internal class representation associated with the object. The internal class representation provides information regarding the Java™ object (e.g., object size, object type, static fields, etc.) As a result, the invention allows quick access to information regarding Java™ objects. This means that the processing time conventionally needed to access information regarding Java™ objects is reduced. Thus, the invention can enhance performance of virtual machines, especially in systems with limited computing power and/or memory.

The invention can be implemented in numerous ways, including as a method, an apparatus, a computer readable medium, and a database system. Several embodiments of the invention are discussed below.

As a Java™ object representation suitable for use by a Java™ virtual machine, one embodiment of the invention includes a first reference to an internal class representation of the Java™ object, a second reference to instance fields associated with the Java™ object. The first reference is a direct reference to the internal class representation of the Java™ object.

As a method for representing a Java™ object in a virtual machine, one embodiment of the invention includes the acts of: allocating a first reference in a memory portion of the virtual machine, wherein the first reference is a reference to an internal class representation of the Java™ object; and allocating a second reference in a memory portion of the virtual machine, wherein the second reference is a reference to instance fields associated with

the Java™ object; and wherein the first reference is a direct reference to the internal class representation of the Java™ object.

As a method of accessing information regarding a Java™ object, one embodiment of the invention includes the acts of identifying an object representation associated with the Java™ object; using a first reference in the object representation to locate an appropriate internal class representation associated with the Java™ object; accessing information regarding the Java™ object from the internal class representation; and wherein the object is represented in a Java™ virtual machine.

As a computer readable media including computer program code for a Java™ object representation suitable for use by a Java™ virtual machine, one embodiment of the invention includes computer program code for a first reference to an internal class representation of the Java™ object; computer program code for a second reference to instance fields associated with the Java™ object. The first reference is a direct reference to the internal class representation of the Java™ object.

These and other aspects and advantages of the present invention will become more apparent when the detailed description below is read in conjunction with the accompanying drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

The present invention will be readily understood by the following detailed description in conjunction with the accompanying drawings, wherein like reference numerals designate like structural elements, and in which:

Fig. 1A shows a progression of a simple piece of a Java™ source code through execution by an interpreter, the Java™ virtual machine.

Fig. 1B illustrates a simplified class file.

Fig. 2 represents a Java™ computing environment including a Java™ object representation in accordance with one embodiment of the invention.

Fig 3 illustrates a method for representing Java™ objects in a Java™ computing environment.